```
/*
 * List.h
 *
 *  Created on: Sep 6, 2016
 *      Author: sanjay
 */

#ifndef LIST_H_
#define LIST_H_

#include <iostream>
using namespace std;

/*
 * struct Node represents a single node in the List
 * It contains previous and next pointerts to neigbouring nodes
 * and val as data
 */
struct Node
{
    Node *prev;
    int val;
    Node *next;

    // Node constructor initialized prev/next to NULL
    // and sets the given value in the node
    Node(int v)
    {
        val = v;
        prev = NULL;
        next = NULL;
    }
};

/*
 * class List represents the entire linked list
 * It holds head and tail pointers to the first and last node in the list
 * It provides the following operations
 * - addToFront()   - Add a value to the beginning of the list
 * - addToBack()    - Add a value to the end of the list
 * - printForward() - Print the list from first node to the last
 * - printBaclward()- Print the list from last node to the first
 * - deleteNode()   - Delete a single node, given the value
 * - deleteTree()   - Delete the entire list
 */
class List
{
private:
    Node *head;
    Node *tail;

public:
    // Constructor initializes the list with NULL head and tail pointers
    List()
    {
        head = tail = NULL;
    }

    // Destructor calls the deleteList to delete all nodes
    virtual ~List()
    {
        deleteList();
    }

    // deleteList walks thru the list from front to back, deleting all nodes
    void deleteList()
    {
        // Pointer to the node AFTER current as we will delete current and
lost addr of next node
        Node *nextNode;

        // Walk thru the list from head to tail
        // As we will delete the current node, we will capture the address of
next node before deleting
        // We will NOT do current=current->next as current is deleted
        // Insted we will do current=nextNode (which is the address of next
node)
        cout << "Deleting the entire list..." << endl;
        for (Node *current = head; current; current = nextNode)
        {
            nextNode = current->next;
            cout << current->val << "\t";
            delete current;
        }
        cout << endl;
    }

    // deleteNode deletes the first matching value found in the list
    bool deleteNode(int val)
    {
        // Iterate thru the entire list
        for (Node *current = head; current; current = current->next)
        {
            // Check if we found the value
            if (current->val == val)
            {
                // Is the the only node?
                // That will be true if head and tail point to the same node
                if (head == tail)
                {
                    // In this case, simply delete node
                    // and set head and tail to NULL
                    // as we have deleted the only node remaining in the list
                    head = tail = NULL;
                    delete current;
                    return true;
                }

                // Ok, there are more than one nodes
                // Are we on the first node?
                // If current points to head, the value was found in head node
                if (current == head)
                {
                    // Move away to the next node
                    // As we are going to delete the head node
                    // Also, the second node will not have any node prev to it
                    // after deletion. So, set the prev of new head to NULL
                    head = head->next;
                    head->prev = NULL;
                    delete current;
                    return true;
                }

                // It was not the only node, nor the first node
                // Are we on the last node?
                // If current points to tail, the value was found in tail node
                if (current == tail)
                {
                    // Move away to the prev node
                    // as we are going the delete the tail node
                    // Also, the second last node will not have any node next
to it

                    // after deletion. So, set the next of new tail to NULL
                    tail = tail->prev;
                    tail->next = NULL;
                    delete current;
                    return true;
                }

                // Finally...
                // It was not the only node
                // We were not on the head node
                // We were not on the tail node
                // So, we are on some middle node, which has nodes prev and
nex to it

                // Connect current prev to next and next to prev
                // and delete current
                current->next->prev = current->prev;
                current->prev->next = current->next;
                delete current;
                return true;
            }
        }
        // We went thru all the nodes and couldn't find the value
        return false;
    }

    // addToFront adds a node before the head node
    bool addToFront(int val)
```

```
{
    // Allocate a new node and set the data
    Node *node = new Node(val);
    if (NULL == node)
        return false;

    // Check if we have a head node or this is the first node
    if (NULL == head)
    {
        // If its the first node, head and tail should point to it
        head = tail = node;
    } else
    {
        // There is alread a head node, so add this before it
        // This node next will point to existing head
        // and then become the new head
        node->next = head;
        head->prev = node;
        head = node;
    }
    return true;
}

// addToBack adds a node after the tail node
bool addToBack(int val)
{
    // Allocate a new node and set the data
    Node *node = new Node(val);
    if (NULL == node)
        return false;

    // Check if we have a head node or this is the first node
    if (NULL == head)
    {
        // If its the first node, head and taill should point to it
        head = tail = node;
    } else
    {
        // There is already a tail node, so add this after it
        // This node prev will point to existing tail
        // and then become the new tail
        node->prev = tail;
        tail->next = node;
        tail = node;
    }
    return true;
}

// printForward will print all nodes from head to tail
void printForward()
{
    for (Node *current = head; current; current = current->next)
        cout << current->val << "\t";
        cout << endl;
    }

    void printBackward()
    {
        for (Node *current = tail; current; current = current->prev)
            cout << current->val << "\t";
        cout << endl;
    }
};

#endif /* LIST_H_ */
```